STRNCAT

Requires careful tracking of character count in the buffer and use of appropriate units, and must be null terminated.

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-23

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 7219 bytes

Attack Category	• Malicious Input • Denial of Service
Vulnerability Category	• Buffer Overflow • Unconditional
Software Context	• String Management
Location	
Description	strncat() requires careful tracking of character count in the buffer and must be null terminated. strncat() concatenates one string onto another, adding no more of the second string than the specified count of characters. There are two basic problems with strncat. First, strncat() uses a max count parameter, not a size parameter, for copying characters. That is, the user must provide the maximum characters to copy, which could overrun the buffer size if there are already too many characters in the buffer. Common misuse of strncat() family functions can result in buffer overrun. The second problem is that many implementations of strncat() do not guarantee that the resulting string will be null terminated. This can happen if the concatenated string is exactly the size of the allowed max count in the buffer, not leaving space for the final null character. The next time the string is accessed, the program will read off the end of the string and potentially cause a DoS or buffer overrun problem. Note: For routines that use wide characters (WCHAR or Unicode), the counts need to be specified as *number of characters*, not *number of bytes*. On Windows, the StringCch replacement functions accept character counts whereas the

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

	StringCb functions accept byte counts. Specifying the count incorrectly can lead to buffer overflow.
APIs	**Function Name** / **Comments**: _mbsnbcat — appends max count *bytes* in a MBC string; _tcsncat; StrCatBuff — uses BuffSize, not "max chars to append"; StrCatBuffA — uses BuffSize, not "max chars to append"; StrCatBuffW — uses BuffSize, not "max chars to append"; StrCatN; StrCatNA; StrCatNW; StrNCat; strncat — have to keep track of size as it builds up; StrNCatA; StrNCatW; wcsncat
Method of Attack	If coded incorrectly, unpredictable behavior may occur even in the absence of an attacker. If attacker can control string to be appended, arbitrary code execution could result.
Exception Criteria	
Solutions	see below

APIs:

Function Name	**Comments**
_mbsnbcat	appends max count *bytes* in a MBC string
_tcsncat	
StrCatBuff	uses BuffSize, not "max chars to append"
StrCatBuffA	uses BuffSize, not "max chars to append"
StrCatBuffW	uses BuffSize, not "max chars to append"
StrCatN	
StrCatNA	
StrCatNW	
StrNCat	
strncat	have to keep track of size as it builds up
StrNCatA	
StrNCatW	
wcsncat	

Solutions:

Solution Applicability	**Solution Description**	**Solution Efficacy**
Anywhere strncat() is used.	There are a variety of solutions: 1) Replace strncat with a function that takes a buffer size (instead of max count) and terminates the string properly. This avoids the common misuse and complication of keeping a "max count	Errors can still occur, but this reduces the likelihood.

	to append" parameter and ensures that the string is null terminated in all cases. For example, use StringCchCatN() or strlcat(), where supported. 2) Replace strncat with a routine like snprintf for generic string formatting capabilities. However, be aware that the snprintf function has some platform-specific issues that need to be understood in order to ensure safe usage.
Signature Details	strncat(*d, *s, int charcount)

Examples of Incorrect Code	`const int BUFSIZE = 18;`
	`char d[BUFSIZE];`
	`char w[] = "Word";`
	`d[0] = '\0';`
	`while (strlen(d) < BUFSIZE-1) {`
	`strncat(d, w, BUFSIZE); /* Results in overflow */`
	`}`

Examples of Corrected Code	`/* if one must use strncat(), the following is safe */`
	
	`const int BUFSIZE = 18;`
	`char d[BUFSIZE];`
	`char w[] = "Word";`
	`d[0] = '\0';`
	`while (strlen(d) < BUFSIZE-1) {`
	`strncat(d, w, BUFSIZE-1-strlen(d)); /* Avoids overflow */`
	`}`
	`d[BUFSIZE-1] = '\0'; /* Ensure null termination */`

	```
	/* on platforms such as MacOS and
	FreeBSD that support it, strlcat()
	is a better choice than strncat()
	*/

	const int BUFSIZE = 18;
	char d[BUFSIZE];
	char w[] = "Word";
	d[0] = '\0';
	while (strlen(d) < BUFSIZE-1) {
	strlcat(d, w, BUFSIZE); /* Avoids
	overflow and null terminates */
	}
	```
Source References	• Howard, Michael & LeBlanc, David C. *Writing Secure Code*, *2nd ed*. Redmond, WA: Microsoft Press, 2002, ISBN: 0735617228. • man page for sprintf() • man page for strlcat()
Recommended Resources	• MSDN on safe string functions.[2]
Discriminant Set	**Operating System**: • Any **Languages**: • C • C++

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com